Yevgeniya Okuneva
Stat 242
HW4

Working with Airline Data in Blocks

Airline Data in Blocks

Introduction

In this assignment, we work with data in blocks. The point is to read a subset of data, process it, discard it and continue with the next subset of the data rather then reading all data into memory and do subsetting on the whole set of data. We will use different approaches on the same data in order to compare the performance. We use UNIX shell tools, connections in R, and SQLite.

The data consists of 22 csv files for years of 1987 to 2008, one for each year. Files are compressed as tar.bz2 into 2 files, one for each decade. In order to enhance the performance, we first uncompress files using UNIX shell command. Next we compute the number of flights leaving each of the four given airports (OAK,LAX, SFO, SMF), mean and standard deviation of the arrival and departure delay times. Details for each of the 3 approaches are provided in subsequent sections.

UNIX shell/ pipe() R

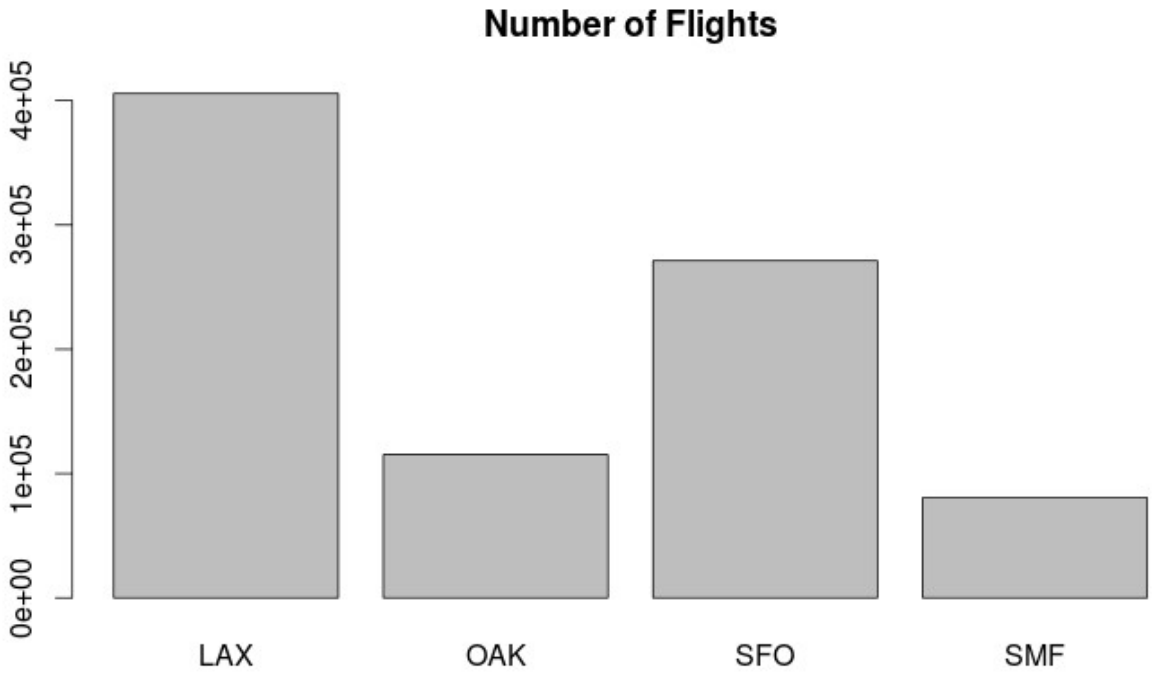
We use “system” and “intern” commands in R in order to call shell command from R. The command cuts files at “,”, pulls the 17th column which has the data for all where the flights were originated. Next we grep the four airports (LAX, OAK, SFO, SMF) and sort the data. Final step is to use “uniq” shell function with “-c” argument to count the number of lines for each of the four airports. Next we format the output in R so that we can plot it.

Table 1 and Figure 1 povide summary for the number of flights originated in each of the four airports.

Table 1. Number of flights originated in LAX, OAK, SFO, and SMF airports

Number of Flights	Airport
405745	LAX
115189	OAK
271195	SFO
80613	SMF

Figure 1. Number of flights originated in LAX, OAK, SFO, and SMF airports



In order to compute mean and standard deviation, we use pipe() to create a connection in R and then read data in blocks using readLines() command. Using shell command, we extract columns with the Arrival Delay time and the origin airport, and grep 4 airports we are interested in. We create a container with all 0's in order to store data we get from blocks. The approach is to get data from the first block, store it in the container, obtain results for the second block and add it to the results from 1st block, then obtain results for 3rd block and add them to the sum from 1st and 2nd blocks, and etc for all subsequent blocks. To be more precise, there are 4 values for each of the 4 airports that we are interested in.

1. Sum of the Arrival Delay time, by airport (missing values are removed)
2. Count of the number of flights, by airport (those with missing Arrival Delay time are removed)
3. Count of the number of flights, by airport including those with missing Arrival Delay
4. Sum of the squared Arrival Delay time, by airport (missing values are removed)

Then 1 and 2 are used to compute mean by simply dividing 1 by 2. Number 3 is used just to see how many flights left each of the 4 airports, in other words, same information as we got from shell script above. To compute variance we use 1, 2, and 4 using the following formula

$$\text{Sum}(x\text{^}2)/n + \text{Xbar}\text{^}2$$

In other words, we take 4, divide it by 2 and add squared mean obtained in the previous step for each of the airports.

Table 2 provides summary of results for obtaining Sums of ArrDelay and ArrDelay squared times and the number of flights for each airport.

Table 2.Summary of computed sums and # of flights

	SFO	LAX	SMF	OAK
Sum of ArrDelay	21085195	23910421	4272059	5767588
# of Flights(missing excluded)	2651722	3980396	797126	1136838
# of Flights	2711958	4057452	806133	1151897
Sum of ArrDelay^2	2600123485	3111081189	524963815	582455010

Table 3 provides means computed for each of the airports.

Table 3. Mean, by airport

	SFO	LAX	SMF	OAK
Mean	7.95151	6.007046	5.359327	5.07336

Table 4 provides standard deviations for each of the airports.

Table 4. Standard deviation, by airport

	SFO	LAX	SMF	OAK
SD	30.28721	27.30415	25.0968	22.05919

We next profile our function for different block sizes.

Below are the partial outputs from Rprof for different block sizes showing how much time and memory(last column) the processes took.

Block size = 100

```
$by.self
                          self.time self.pct total.time total.pct mem.total
"strsplit"                    12.48    12.54      12.48     12.54    2636.1
"data.frame"                   8.08     8.12      71.48     71.80   14933.0
"deparse"                      5.20     5.22      12.70     12.76    2631.7
"factor"                       4.34     4.36      22.24     22.34    4608.7
"match"                        3.96     3.98       7.74      7.77    1645.6
"as.list"                      3.72     3.74       5.16      5.18    1073.1
"sort.list"                    3.68     3.70      10.70     10.75    2228.8
".deparseOpts"                 2.96     2.97       4.64      4.66     956.7
"unique"                       2.94     2.95       5.14      5.16    1050.4
"as.data.frame"                2.50     2.51      32.64     32.78    6739.3
"make.names"                   2.48     2.49       2.86      2.87     609.4



$sample.interval
[1] 0.02

$sampling.time
[1] 99.8
```

Block size = 1000

```
$by.self
                          self.time self.pct total.time total.pct mem.total
"strsplit"                    27.76    38.36      27.76     38.36    2992.4
"readLines"                    5.40     7.46       5.40      7.46     598.9
"factor"                       3.48     4.81      11.88     16.42    1203.5
"data.frame"                   2.76     3.81      53.94     74.54    5729.2
"match"                        2.58     3.57       4.16      5.75     421.3
"deparse"                      2.50     3.45       5.22      7.21     541.5
"sort.list"                    2.36     3.26       4.10      5.67     418.4
"unlist"                       1.66     2.29      29.80     41.18    3205.6
"unique.default"               1.64     2.27       1.74      2.40     178.5
".deparseOpts"                 1.26     1.74       1.68      2.32     165.7
"make.names"                   1.24     1.71       1.28      1.77     135.7



$sample.interval
[1] 0.02

$sampling.time
[1] 72.4
```

Block size = 10 000

```
$by.self
                          self.time self.pct total.time total.pct mem.total
"strsplit"                    27.84    59.54      27.84     59.54    1063.3
"readLines"                    5.58    11.93       5.58     11.93     261.7
"factor"                       2.46     5.26       5.64     12.06     208.2
"as.numeric"                   1.52     3.25       1.68      3.59      63.8
"match"                        1.30     2.78       1.50      3.21      54.1
"unlist"                       1.14     2.44      28.98     61.98    1099.5
"unique.default"               1.04     2.22       1.08      2.31      38.0
"is.factor"                    0.82     1.75      17.82     38.11     659.2
"sort.list"                    0.60     1.28       0.82      1.75      24.4
"data.frame"                   0.48     1.03      35.48     75.88    1337.7
"matrix"                       0.36     0.77      29.18     62.40    1103.6

$sample.interval
[1] 0.02

$sampling.time
[1] 46.76
```

Block size = 100 000

```
$by.self
                        self.time self.pct total.time total.pct mem.total
"strsplit"                  27.36    57.53      27.36     57.53     924.6
"readLines"                  5.50    11.56       5.50     11.56      61.3
"factor"                     3.10     6.52       6.96     14.63      84.8
"match"                      1.92     4.04       1.94      4.08      24.9
"unique.default"             1.80     3.78       1.80      3.78      27.0
"unlist"                     1.66     3.49      29.02     61.02     967.0
"as.numeric"                 1.36     2.86       1.42      2.99       9.5
"matrix"                     1.14     2.40      30.08     63.25     982.5
"is.factor"                  1.00     2.10      17.96     37.76     528.7
"tapply"                     0.60     1.26      41.96     88.23    1126.8
"t.default"                  0.60     1.26       0.60      1.26       7.0

$sample.interval
[1] 0.02

$sampling.time
[1] 47.56
```

Block size = 1 000 000

```
$by.self
                        self.time self.pct total.time total.pct mem.total
"strsplit"                 28.64    61.12      28.64     61.12     978.1
"readLines"                 5.28    11.27       5.28     11.27       4.3
"factor"                    2.80     5.98       5.34     11.40     102.0
"unlist"                    1.74     3.71      30.38     64.83    1020.3
"match"                     1.40     2.99       1.40      2.99      27.5
"as.numeric"                1.22     2.60       1.24      2.65      12.0
"t.default"                 1.18     2.52       1.18      2.52      28.6
"unique.default"            1.14     2.43       1.14      2.43      57.1
"matrix"                    1.08     2.30      31.36     66.92    1041.6
"is.factor"                 0.78     1.66      18.80     40.12     625.5
"as.data.frame.matrix"      0.46     0.98       3.48      7.43     117.9

$sample.interval
[1] 0.02

$sampling.time
[1] 46.86
```

Block Size = All lines

```
$by.self
                        self.time self.pct total.time total.pct mem.total
"strsplit"                 45.86    71.34      45.86     71.34     981.8
"readLines"                 6.36     9.89       6.36      9.89      24.4
"factor"                    2.76     4.29       5.48      8.53     118.8
"unlist"                    1.80     2.80      47.66     74.14    1023.8
"unique.default"            1.50     2.33       1.50      2.33      64.7
"matrix"                    1.26     1.96      48.76     75.86    1048.8
"as.numeric"                1.24     1.93       1.24      1.93       8.3
"match"                     1.22     1.90       1.22      1.90      33.3
"is.factor"                 0.92     1.43      28.54     44.40     614.7
"t.default"                 0.50     0.78       0.50      0.78      16.6


$sample.interval
[1] 0.02

$sampling.time
[1] 64.28
```

We see that the block size has a big effect on performance and memory when reading data into R from pipe connection where we first strip off all unneed data by extracting only the columns we are interested in and then extracting airports that we are interested in. Reading too few lines (i.e. 100) significantly slows the performanceThe amount of total memory is a little bit confusing. It also show a similar pattern as the performance time. So, when we only extract the data we need and then pipe it into R it doesn't matter how big are the blocks, even the full data set gets processed pretty fast unless we read data in very small blocks.

SQLite

We first create a data base containing data for all 22 years (see attached create.SQlite3.database.txt script).

Next, the commands to get the average, count, and variance for 4 airports, by airport where called from R using Sqlite package for interfacing Sqlite 3 with R. Next once the connection was established, we used fetch on the whole data base to obtain average, count, and variance. Weirdly, there is not build in function in SQL for variance or standard deviation. So, we used the same formula as in previous sections to find variance when looked at the whole data base at once. Interestingly, SQRT command didn't work either, so we had to obtain variance and then take a square root of it outside of SQL query. Note: there is a way to write Variance function in SQL but for the purpose of this assignment it was easier to use the above approach.

Table 5 provides summary of results, by airport. Note that the computed values are slightly different which is most likely due to NAs handling, or to be more precise SQL's ignorance of missing values. One option to fix that is to specify ArrDealy IS NOT NULL; however, it did not work. The other approach is to find all missing values and set them to some large negative number (i.e. -99999) and then select all values greater then that.

Table 5. Results obtained from SQL query

Origin	mean	count	variance	sd
LAX	5.892965	4057452	732.0303	27.05606
OAK	5.007034	1151897	480.5781	21.92209
SFO	7.774897	2711958	898.3134	29.97188
SMF	5.299447	806133	623.1283	24.96254

Profiling showed great performance when we ran the command through the whole data base rather then doing computations in blocks

```
$by.self
         self.time self.pct total.time total.pct mem.total
".Call"     88.92      100      88.92       100         0

$by.total
                   total.time total.pct mem.total self.time self.pct
".Call"                 88.92       100         0     88.92      100
"fetch"                 88.92       100         0      0.00        0
"is"                    88.92       100         0      0.00        0
"sqliteFetch"           88.92       100         0      0.00        0
"standardGeneric"       88.92       100         0      0.00        0
".valueClassTest"       88.92       100         0      0.00        0

$sample.interval
[1] 0.02

$sampling.time
[1] 88.92
```

Package

Was not created as most functions are not general.

GitHut Repository

Can be found at https://github.com/yokuneva/HW4.stat242